



Indian Institute of Technology Jodhpur

Fundamentals of Distributed Systems

Assignment – 1-solved

Name: Sanjay kumar ranjan

Submission Deadline: 25 June 2025

1. Vector Clocks and Causal Ordering

[10 Marks]

Objective

To move beyond simple event ordering by implementing **Vector Clocks** to capture the causal relationships between events in a distributed system. You will apply this to build a causally consistent, multi-node key-value store.

Problem Description

You will build a distributed key-value store with three or more nodes. The key challenge is to ensure that writes to the store are **causally ordered**. If event B is causally dependent on event A (e.g., a value is read and then updated), all nodes must process event A before they process event B. Simple Lamport clocks are insufficient for this, so you will use Vector Clocks.

Technology Constraints

- **Programming Language:** The entire application logic for the nodes and client must be written exclusively in **Python**.
- **Containerization:** The system must be containerized and orchestrated solely using **Docker** and **Docker Compose**.

Tasks

Your implementation should cover the following tasks:

1. **Node Implementation with Vector Clocks:** Create a Python script for a node. Each node must maintain its own local key-value data and a Vector Clock.
2. **Vector Clock Logic:** Implement the rules for incrementing the clock on local events, including the clock in sent messages, and updating the local clock upon receiving a message.
3. **Causal Write Propagation:** Implement the Causal Delivery Rule. When a node receives a replicated write message, it must delay processing that write until the causal dependencies are met by checking the message's vector clock against its own. Messages that cannot be delivered must be buffered.

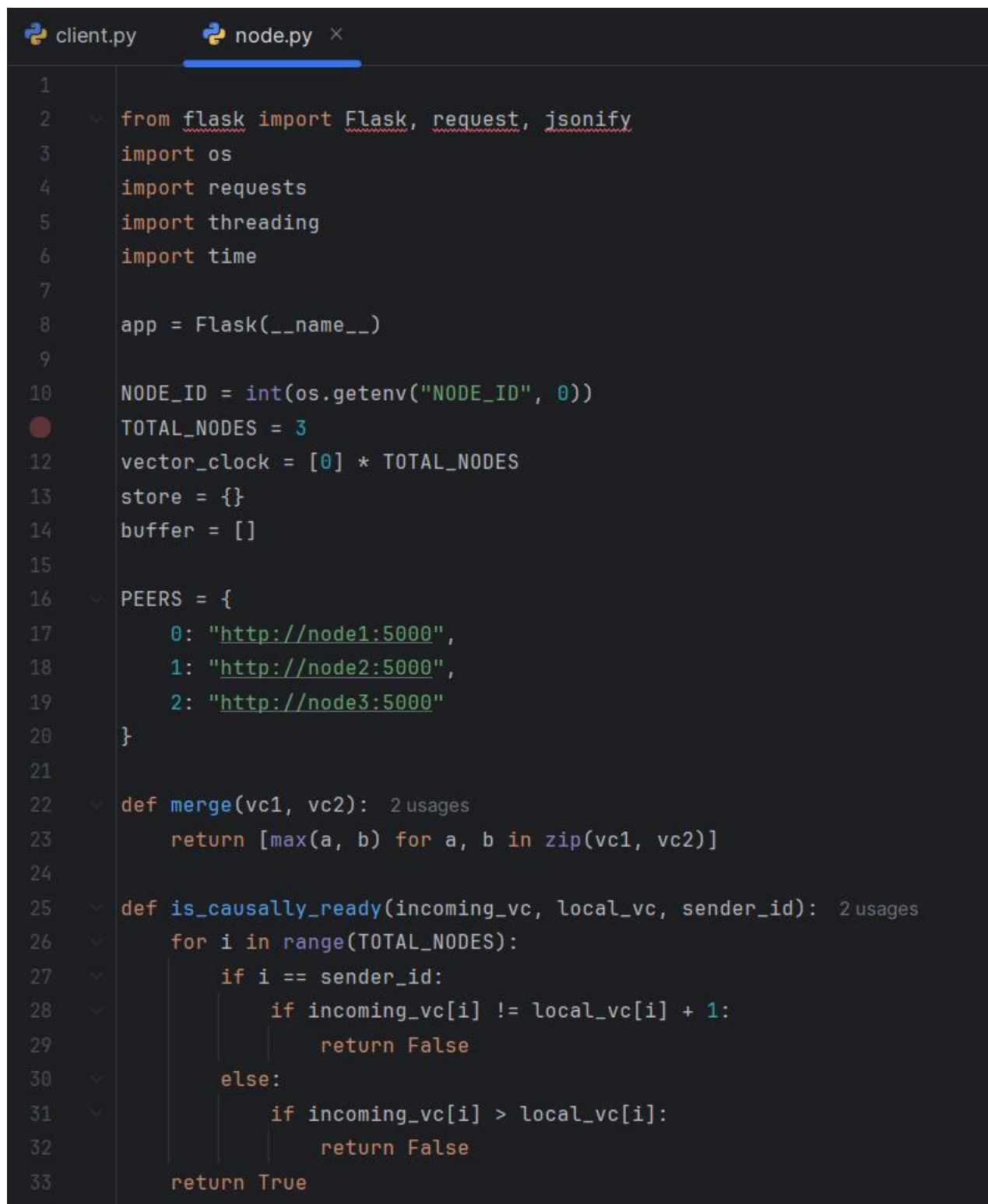
Here,

node.py contains Key-value store with vector clock logic

Client.py Sends reads/writes to different nodes

Dockerfile is for containerizing node.py

docker-compose.yml is for setting up 3-node system with networking



```
1
2  from flask import Flask, request, jsonify
3  import os
4  import requests
5  import threading
6  import time
7
8  app = Flask(__name__)
9
10  NODE_ID = int(os.getenv("NODE_ID", 0))
11  TOTAL_NODES = 3
12  vector_clock = [0] * TOTAL_NODES
13  store = {}
14  buffer = []
15
16  PEERS = {
17      0: "http://node1:5000",
18      1: "http://node2:5000",
19      2: "http://node3:5000"
20  }
21
22  def merge(vc1, vc2): 2 usages
23      return [max(a, b) for a, b in zip(vc1, vc2)]
24
25  def is_causally_ready(incoming_vc, local_vc, sender_id): 2 usages
26      for i in range(TOTAL_NODES):
27          if i == sender_id:
28              if incoming_vc[i] != local_vc[i] + 1:
29                  return False
30          else:
31              if incoming_vc[i] > local_vc[i]:
32                  return False
33      return True
```

```

35 @app.route('/put', methods=['POST'])
36 def put():
37     global vector_clock
38     data = request.json
39     key = data['key']
40     value = data['value']
41
42     vector_clock[NODE_ID] += 1
43     store[key] = (value, vector_clock[:])
44
45     for peer_id, peer_url in PEERS.items():
46         if peer_id != NODE_ID:
47             try:
48                 requests.post(url=f"{peer_url}/replicate", json={
49                     "key": key,
50                     "value": value,
51                     "vc": vector_clock[:],
52                     "sender_id": NODE_ID
53                 })
54             except Exception as e:
55                 print(f"Failed to replicate to node {peer_id}: {e}")
56
57     return jsonify({"msg": "Value stored with causal consistency", "vc": vector_clock})

```

```

59 @app.route('/replicate', methods=['POST'])
60 def replicate():
61     global store, vector_clock
62     data = request.json
63     key, value, vc, sender_id = data["key"], data["value"], data["vc"], data["sender_id"]
64
65     if is_causally_ready(vc, vector_clock, sender_id):
66         vector_clock = merge(vector_clock, vc)
67         store[key] = (value, vc)
68         return jsonify({"msg": "Replicated"})
69     else:
70         buffer.append(data)
71         return jsonify({"msg": "Buffered due to causal dependency"})
72
73 @app.route('/get', methods=['GET']) 1 usage (1 dynamic)
74 def get():
75     key = request.args.get('key')
76     if key in store:
77         return jsonify({"key": key, "value": store[key][0], "vc": store[key][1]})
78     return jsonify({"msg": "Key not found"}), 404

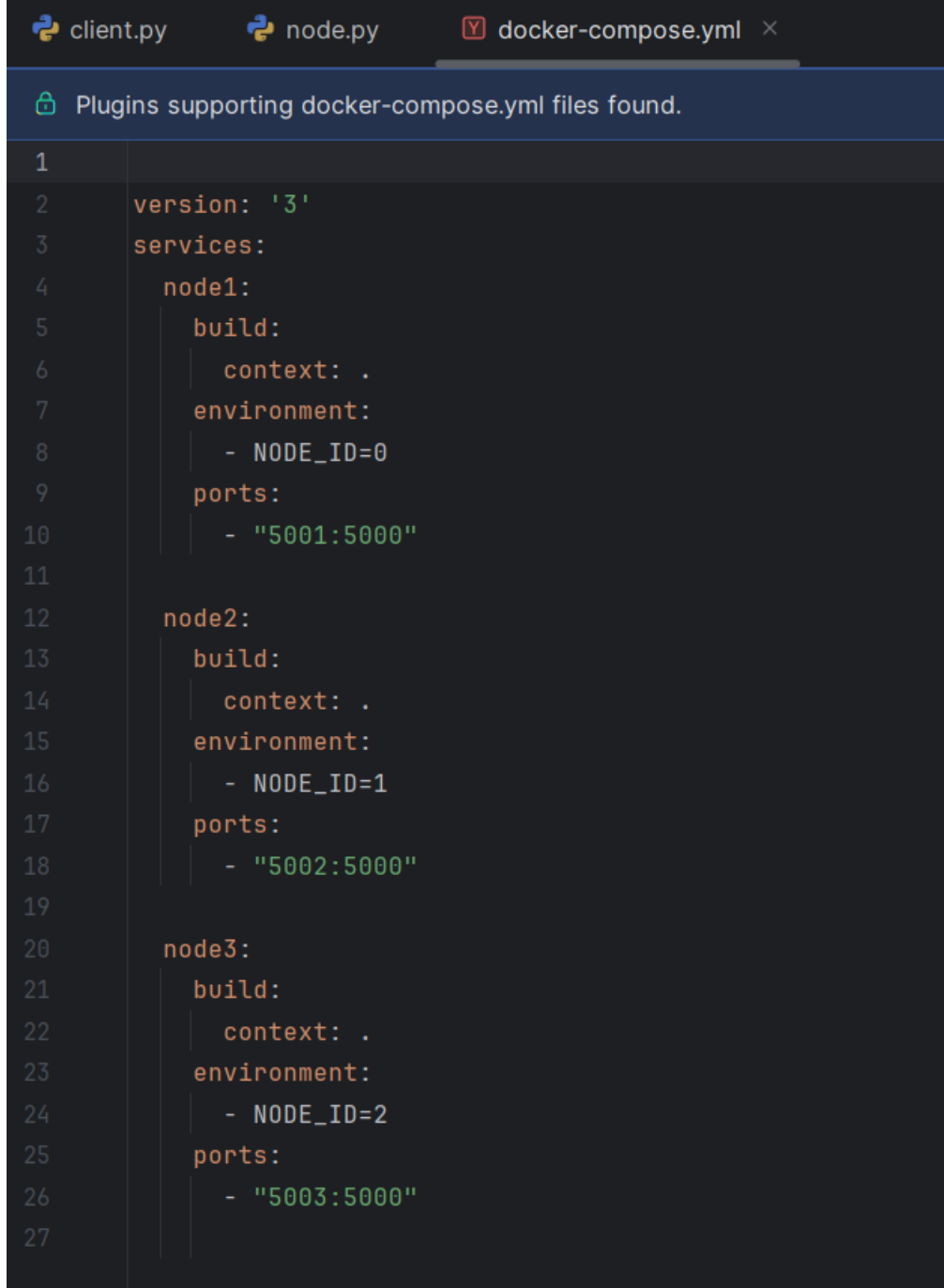
```

```

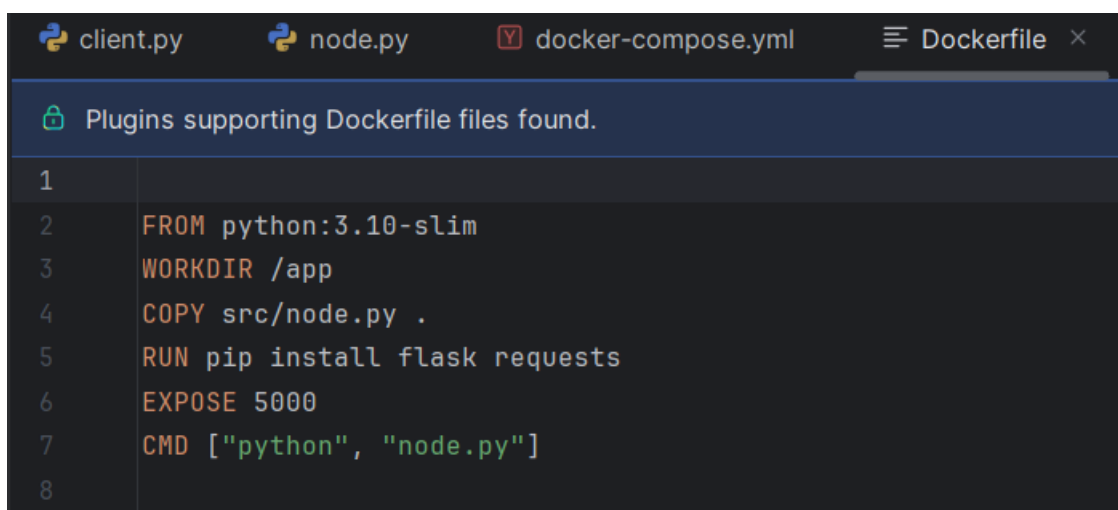
80 @app.route('/status', methods=['GET'])
81 def status():
82     return jsonify({
83         "store": store,
84         "vc": vector_clock,
85         "buffered_msgs": len(buffer)
86     })
87
88 def buffer_check(): 1 usage
89     while True:
90         for data in buffer[:]:
91             if is_causally_ready(data["vc"], vector_clock, data["sender_id"]):
92                 store[data["key"]] = (data["value"], data["vc"])
93                 vector_clock = merge(vector_clock, data["vc"])
94                 buffer.remove(data)
95             time.sleep(1)
96
97 if __name__ == '__main__':
98     threading.Thread(target=buffer_check, daemon=True).start()
99     app.run(host='0.0.0.0', port=5000)
100

```

4. **Containerization and Networking:** Write a 'Dockerfile' for your node and a 'docker-compose.yml' file to run a 3-node system.



```
1
2 version: '3'
3 services:
4   node1:
5     build:
6       context: .
7     environment:
8       - NODE_ID=0
9     ports:
10      - "5001:5000"
11
12   node2:
13     build:
14       context: .
15     environment:
16       - NODE_ID=1
17     ports:
18      - "5002:5000"
19
20   node3:
21     build:
22       context: .
23     environment:
24       - NODE_ID=2
25     ports:
26      - "5003:5000"
27
```



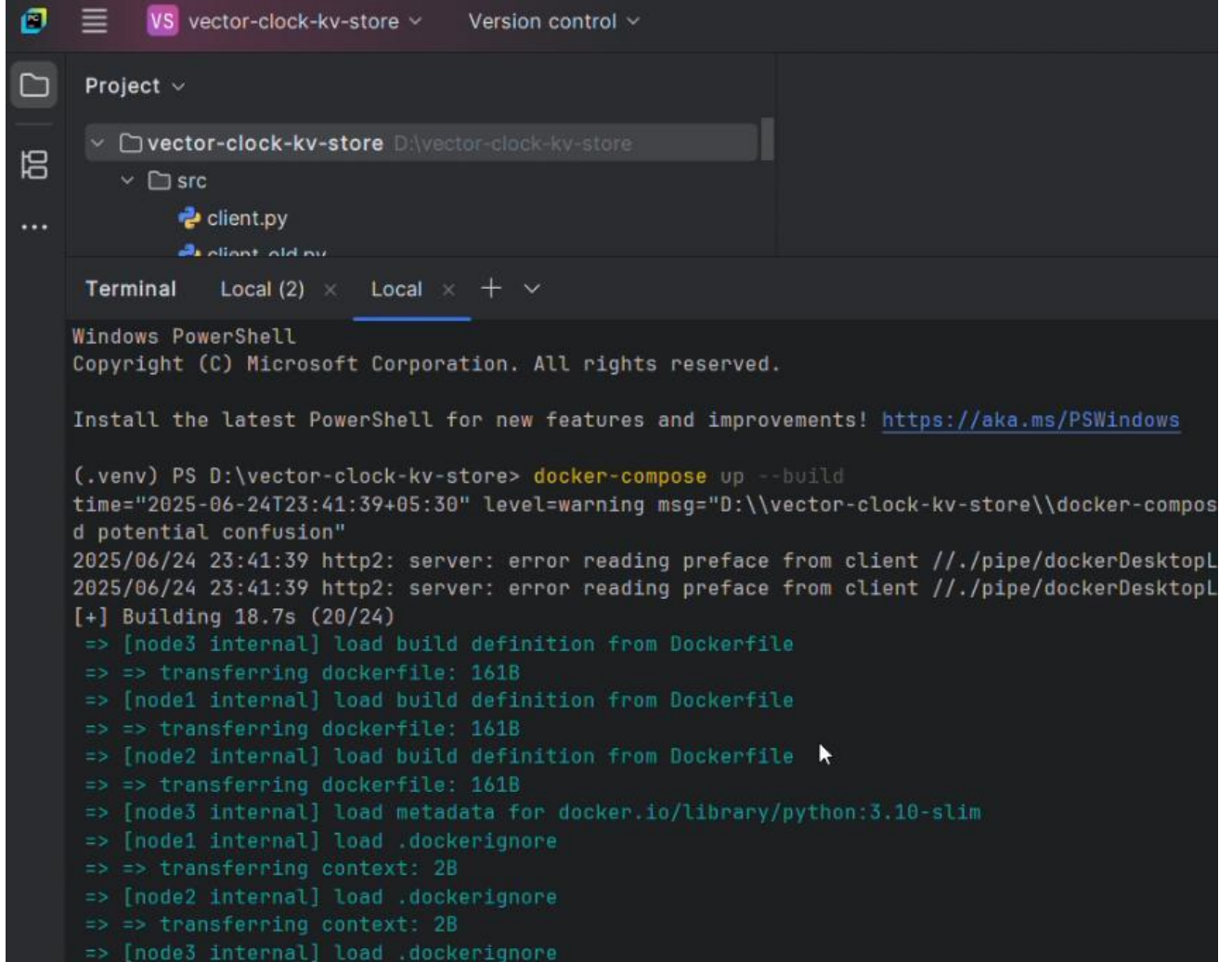
```
1
2 FROM python:3.10-slim
3 WORKDIR /app
4 COPY src/node.py .
5 RUN pip install flask requests
6 EXPOSE 5000
7 CMD ["python", "node.py"]
8
```

5. **Verification and Scenario Testing:** Create a client script and a specific test scenario

to prove that your system maintains causal consistency, even when messages arrive out of order.

```
client.py x node.py docker-compose.yml Dockerfile
1 import requests
2 import time
3
4 nodes = {
5     "node1": "http://localhost:5001",
6     "node2": "http://localhost:5002",
7     "node3": "http://localhost:5003"
8 }
9
10 print("\n🟢 Step 1: Node1 writes x=15")
11 resp = requests.post(url=f"{nodes['node1']}/put", json={"key": "x", "value": "15"})
12 print("Node1 put x=15:", resp.json())
13 time.sleep(2)
14
15 print("\n🟢 Step 2: Node2 writes x=30 (depends on previous write)")
16 resp = requests.post(url=f"{nodes['node2']}/put", json={"key": "x", "value": "30"})
17 print("Node2 put x=30:", resp.json())
18 time.sleep(2)
19
20 print("\n🟢 Step 3: Node3 writes y=90 (independent write)")
21 resp = requests.post(url=f"{nodes['node3']}/put", json={"key": "y", "value": "90"})
22 print("Node3 put y=90:", resp.json())
23 time.sleep(2)
24
25 print("\n🟡 Step 4: Read 'x' and 'y' from all nodes")
26 for name, url in nodes.items():
27     try:
28         x = requests.get(url=f"{url}/get", params={"key": "x"}).json()
29         y = requests.get(url=f"{url}/get", params={"key": "y"}).json()
30         print(f"{name} stores:\n x: {x}\n y: {y}")
31     except Exception as e:
32         print(f"{name} failed to respond:", str(e))
33
```

Building Using Docker Compose



The image shows a VS Code editor window with a project named 'vector-clock-kv-store' open. The file explorer on the left shows the project structure with a 'src' folder containing 'client.py' and 'client_old.py'. The terminal window at the bottom shows the output of the command 'docker-compose up --build' in a Windows PowerShell environment. The output indicates that Docker Compose is building the project, showing progress for three nodes (node1, node2, node3) as they load build definitions and context from the Dockerfile. The build is currently at 18.7s out of 20/24.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS D:\vector-clock-kv-store> docker-compose up --build
time="2025-06-24T23:41:39+05:30" level=warning msg="D:\\vector-clock-kv-store\\docker-compos
d potential confusion"
2025/06/24 23:41:39 http2: server: error reading preface from client //./pipe/dockerDesktopL
2025/06/24 23:41:39 http2: server: error reading preface from client //./pipe/dockerDesktopL
[+] Building 18.7s (20/24)
=> [node3 internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [node1 internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [node2 internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [node3 internal] load metadata for docker.io/library/python:3.10-slim
=> [node1 internal] load .dockerignore
=> => transferring context: 2B
=> [node2 internal] load .dockerignore
=> => transferring context: 2B
=> [node3 internal] load .dockerignore
```

Containers got created


```
=> [node3] exporting to image
=> => exporting layers
=> => writing image sha256:8b804d973784f5f7a3cab77475f079f60decfe95fbc6
=> => naming to docker.io/library/vector-clock-kv-store-node3
=> [node3] resolving provenance for metadata file
=> [node2] resolving provenance for metadata file
=> [node1] resolving provenance for metadata file
[+] Running 4/4
✓ Network vector-clock-kv-store_default Created
✓ Container vector-clock-kv-store-node2-1 Created
✓ Container vector-clock-kv-store-node3-1 Created
✓ Container vector-clock-kv-store-node1-1 Created
Attaching to node1-1, node2-1, node3-1
node3-1 | * Serving Flask app 'node'
node3-1 | * Debug mode: off
node1-1 | * Serving Flask app 'node'
node2-1 | * Serving Flask app 'node'
node3-1 | WARNING: This is a development server. Do not use it in a pro
node1-1 | * Debug mode: off
node2-1 | * Debug mode: off
node3-1 | * Running on all addresses (0.0.0.0)
node1-1 | WARNING: This is a development server. Do not use it in a pro
node2-1 | WARNING: This is a development server. Do not use it in a pro
node3-1 | * Running on http://127.0.0.1:5000
node1-1 | * Running on all addresses (0.0.0.0)
node2-1 | * Running on all addresses (0.0.0.0)
node3-1 | * Running on http://172.20.0.4:5000
node1-1 | * Running on http://127.0.0.1:5000
node2-1 | * Running on http://127.0.0.1:5000
node3-1 | Press CTRL+C to quit
node1-1 | * Running on http://172.20.0.3:5000
node2-1 | * Running on http://172.20.0.2:5000
node1-1 | Press CTRL+C to quit
node2-1 | Press CTRL+C to quit
```

Getting the desired output on the terminal.

```
Terminal Local (2) × Local × Local (3) ×
node2-1 | Press CTRL+C to quit
node3-1 | * Running on http://172.20.0.3:5000
node1-1 | * Running on http://172.20.0.2:5000
node3-1 | Press CTRL+C to quit
node1-1 | Press CTRL+C to quit
node2-1 | 172.20.0.2 - - [25/Jun/2025 17:08:14] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.20.0.2 - - [25/Jun/2025 17:08:14] "POST /replicate HTTP/1.1" 200 -
node1-1 | 172.20.0.1 - - [25/Jun/2025 17:08:14] "POST /put HTTP/1.1" 200 -
node1-1 | 172.20.0.4 - - [25/Jun/2025 17:08:16] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.20.0.4 - - [25/Jun/2025 17:08:16] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.20.0.1 - - [25/Jun/2025 17:08:16] "POST /put HTTP/1.1" 200 -
node1-1 | 172.20.0.3 - - [25/Jun/2025 17:08:18] "POST /replicate HTTP/1.1" 200 -
node2-1 | 172.20.0.3 - - [25/Jun/2025 17:08:18] "POST /replicate HTTP/1.1" 200 -
node3-1 | 172.20.0.1 - - [25/Jun/2025 17:08:18] "POST /put HTTP/1.1" 200 -
node1-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=x HTTP/1.1" 200 -
node1-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=y HTTP/1.1" 200 -
node2-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=x HTTP/1.1" 200 -
node2-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=y HTTP/1.1" 200 -
node3-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=x HTTP/1.1" 200 -
node3-1 | 172.20.0.1 - - [25/Jun/2025 17:08:20] "GET /get?key=y HTTP/1.1" 200 -
```

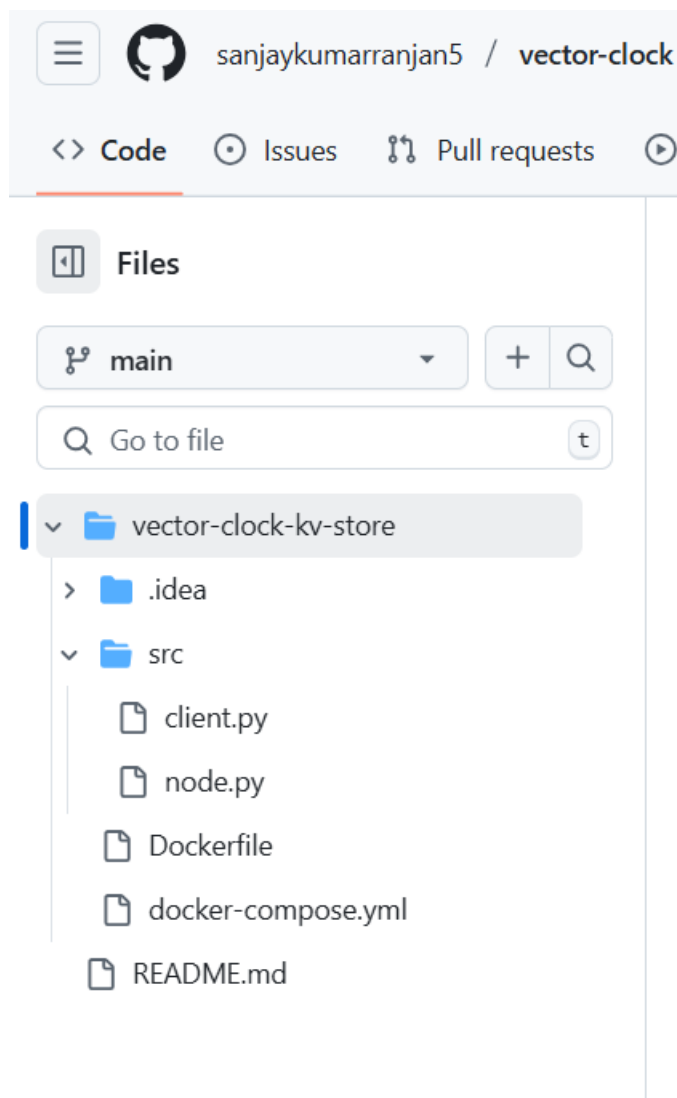
Deliverables

Your final submission must be a single Git repository containing all the required files organized in the exact structure shown below.

[vector-clock/vector-clock-kv-store at main · sanjaykumarranjan5/vector-clock](https://github.com/sanjaykumarranjan5/vector-clock/tree/main/vector-clock-kv-store)

<https://github.com/sanjaykumarranjan5/vector-clock/tree/main/vector-clock-kv-store>

Folder and File Structure



9

Project Report Requirements

You must submit a thorough **Project Report** that details your work.

- **Format:** The report must be a typed document (e.g., created in Microsoft Word, Google Docs, or L^AT_EX) and submitted as a single 'project.report.pdf' file. **Handwritten reports will not be accepted.**

- **Content:** The report should detail your architecture and implementation and use logs with screenshots to prove that your causal consistency test works correctly.
- **Video Link:** The report must include a publicly accessible link to a short (max 3-minute) video demonstrating the project.

<https://youtu.be/ep-hMSYqLxU>

2. Dynamic Load Balancing for a Smart Grid

[10 Marks]

Objective

To design and build a scalable system for a Smart Grid that dynamically balances Electric Vehicle (EV) charging requests across multiple substations based on their real-time load, complete with a comprehensive observability stack.

Problem Description

You will build a system that simulates a Smart Grid managing charging requests from a fleet of EVs. The primary challenge is to prevent overloading any single charging substation. The system must intelligently distribute incoming charging requests to the **least loaded** substation, ensuring grid stability and efficient resource usage. You will use an industry-standard monitoring stack to measure and visualize key performance indicators.

Technology Constraints

- **Programming Language:** All custom services must be written exclusively in **Python**.
- **Containerization & Orchestration:** The entire system must be defined, configured, and run using **Docker** and **Docker Compose**.

Tasks

Your implementation should cover the following tasks:

1. **Microservice Development:** Create two services: a 'charge request service' as the public entry point and a 'substation service' that simulates charging. Instrument the substation to expose its current load as a Prometheus metric.
2. **Custom Dynamic Load Balancer:** Build a new service that acts as the grid's core logic. It must periodically poll the '/metrics' endpoint of each substation to get its current load and use this data to decide where to route new requests.
3. **Observability Stack:** Configure Prometheus to scrape the substation metrics and Grafana to visualize the load on a dashboard.
4. **Containerization & Orchestration:** Write 'Dockerfile's for all services and a 'docker-compose.yml' file to run the entire system, including multiple replicas of the substation service.
5. **Load Testing and Analysis:** Create a Python script to simulate a "rush hour" of EV charging requests and analyze the system's response on your Grafana dashboard.

Deliverables

Your final submission must be a single Git repository containing all the required files organized in the exact structure shown below.

Folder and File Structure

← ↻ <https://github.com/sanjaykumarranjan5/smart-grid-load-l>

Files

main

+ 🔍

Go to file t

> .idea

▼ charge_request_service

Dockerfile

main.py

▼ load_balancer

Dockerfile

main.py

▼ load_tester

test.py

▼ monitoring

▼ grafana

> plugins

dashboard.json

grafana.db

▼ prometheus

prometheus.yml

▼ substation_service


Dockerfile

main.py

README.md

docker-compose.yml

smart-grid-load-l

 sanjaykumarranjan5

Name
..
Dockerfile
main.py

12

Project Report Requirements

You must submit a thorough **Project Report** that details your work.

- **Format:** The report must be a typed document and submitted as a single project_report.pdf file. **Handwritten reports will not be accepted.**
- **Content:** The report should detail your architecture and analyze the system's performance using screenshots from your Grafana dashboard during the load test.
- **Video Link:** The report must include a publicly accessible link to a short (max 3-minute) video demonstrating the project in action.

Github:

[sanjaykumarranjan5/smart-grid-load-balancer](https://github.com/sanjaykumarranjan5/smart-grid-load-balancer)

<https://github.com/sanjaykumarranjan5/smart-grid-load-balancer/tree/main>

Youtube:

1. https://youtu.be/m5_pGoCDxd0

Services to Implement (in Python):

- charge_request_service: Public entry point that receives EV charge requests.
- load_balancer: Core logic of the system. This service must:
 - Periodically scrape load metrics from all substations.
 - Route new charge requests to the **least loaded** substation.
- substation_service: Simulates an EV charging substation.
 - Must expose current load as a **Prometheus-compatible** metric (e.g., on /metrics endpoint).
- load_tester: Simulates "rush hour" by firing many EV charge requests.

Observability Requirements:

- Set up **Prometheus** to scrape metrics from substation services.
- Use **Grafana** to build a dashboard showing substation loads over time.

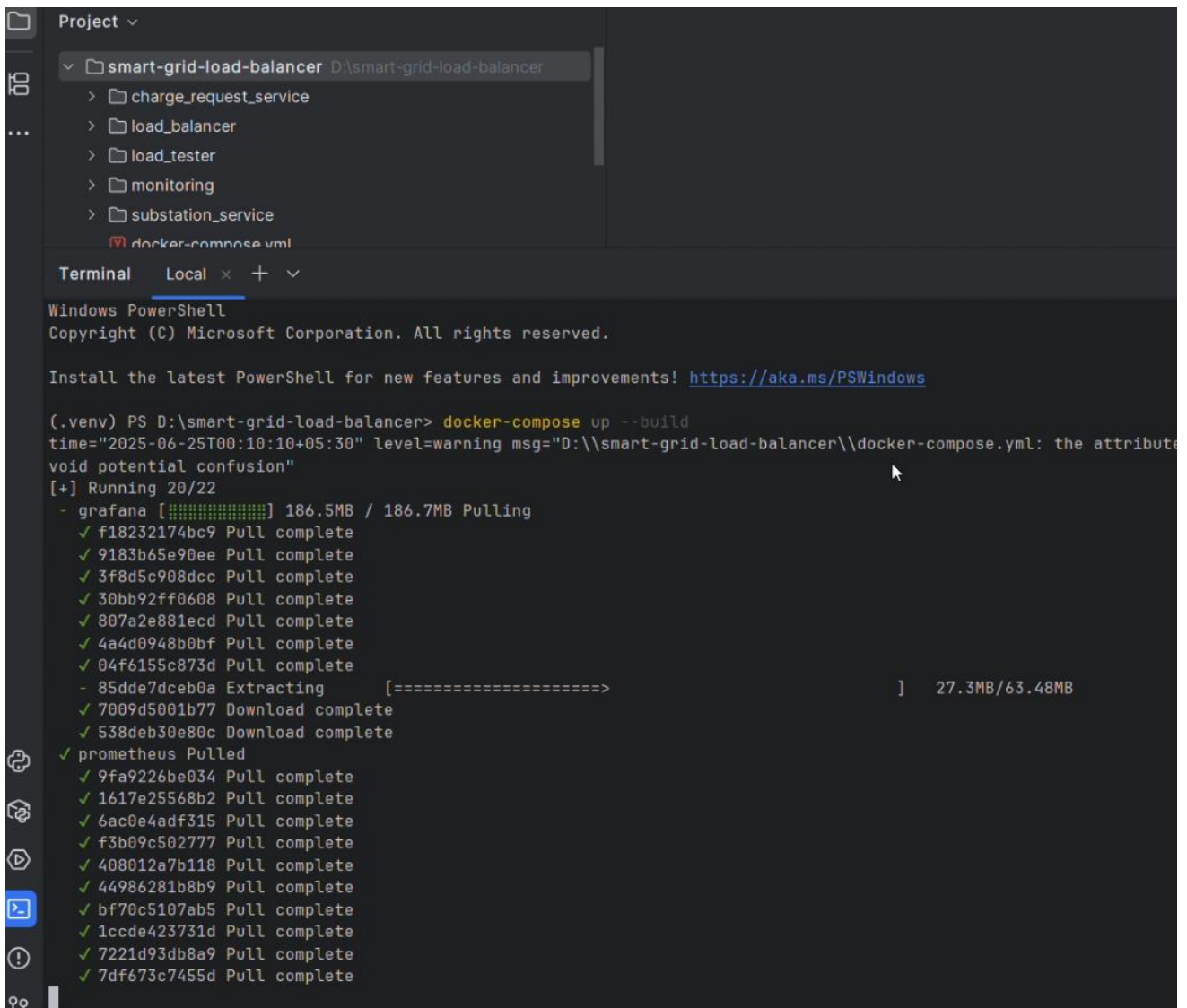
Technical Constraints & Tools:

- Language: Python only for all services
- Containerization: Use Docker and Docker Compose to define and run

everything

- Monitoring: Use Prometheus and Grafana
- Testing: Simulate heavy traffic using load_tester/test.py

Steps: Creating containers using docker compose:



```
Project ▾
  ▾ smart-grid-load-balancer D:\smart-grid-load-balancer
    > charge_request_service
    > load_balancer
    > load_tester
    > monitoring
    > substation_service
    > docker-compose.yml

Terminal Local × + ▾
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS D:\smart-grid-load-balancer> docker-compose up --build
time="2025-06-25T00:10:10+05:30" level=warning msg="D:\\smart-grid-load-balancer\\docker-compose.yml: the attribute
void potential confusion"
[+] Running 20/22
- grafana [#####] 186.5MB / 186.7MB Pulling
  ✓ f18232174bc9 Pull complete
  ✓ 9183b65e90ee Pull complete
  ✓ 3f8d5c908dcc Pull complete
  ✓ 30bb92ff0608 Pull complete
  ✓ 807a2e881ecd Pull complete
  ✓ 4a4d0948b0bf Pull complete
  ✓ 04f6155c873d Pull complete
- 85dde7dceb0a Extracting [=====] 27.3MB/63.48MB
  ✓ 7009d5001b77 Download complete
  ✓ 538deb30e80c Download complete
✓ prometheus Pulled
  ✓ 9fa9226be034 Pull complete
  ✓ 1617e25568b2 Pull complete
  ✓ 6ac0e4adf315 Pull complete
  ✓ f3b09c502777 Pull complete
  ✓ 408012a7b118 Pull complete
  ✓ 44986281b8b9 Pull complete
  ✓ bf70c5107ab5 Pull complete
  ✓ 1ccde423731d Pull complete
  ✓ 7221d93db8a9 Pull complete
  ✓ 7df673c7455d Pull complete
```



```
[+] Running 7/6
✓ Network smart-grid-load-balancer_default Created
✓ Container smart-grid-load-balancer-grafana-1 Created
✓ Container smart-grid-load-balancer-substation2-1 Created
✓ Container smart-grid-load-balancer-prometheus-1 Created
✓ Container smart-grid-load-balancer-substation1-1 Created
✓ Container smart-grid-load-balancer-load_balancer-1 Created
✓ Container smart-grid-load-balancer-charge_request-1 Created
Attaching to charge_request-1, grafana-1, load_balancer-1, prometheus-1, subs
```

Running test.py

The screenshot shows the Visual Studio Code interface. The 'Project' sidebar on the left displays the file structure of the 'smart-grid-load-balancer' project, with 'test.py' selected under the 'load_tester' folder. The main editor window shows the code in 'test.py', which imports the 'requests' library and sends 23 POST requests to 'http://localhost:6000/request'. The 'Run' panel at the bottom displays the output of these requests, all returning a status of 'Charging started'.

```
1
2 > import ...
4
5 for i in range(23):
6     response = requests.post("http://localhost:6000/request_
7     print(f"Request {i+1}: {response.json()}")
8     time.sleep(0.2)
9
```

```
Request 2: {'status': 'Charging started'}
Request 3: {'status': 'Charging started'}
Request 4: {'status': 'Charging started'}
Request 5: {'status': 'Charging started'}
Request 6: {'status': 'Charging started'}
Request 7: {'status': 'Charging started'}
Request 8: {'status': 'Charging started'}
Request 9: {'status': 'Charging started'}
Request 10: {'status': 'Charging started'}
Request 11: {'status': 'Charging started'}
Request 12: {'status': 'Charging started'}
Request 13: {'status': 'Charging started'}
Request 14: {'status': 'Charging started'}
Request 15: {'status': 'Charging started'}
Request 16: {'status': 'Charging started'}
Request 17: {'status': 'Charging started'}
Request 18: {'status': 'Charging started'}
Request 19: {'status': 'Charging started'}
Request 20: {'status': 'Charging started'}
Request 21: {'status': 'Charging started'}
Request 22: {'status': 'Charging started'}
Request 23: {'status': 'Charging started'}
```

Output:

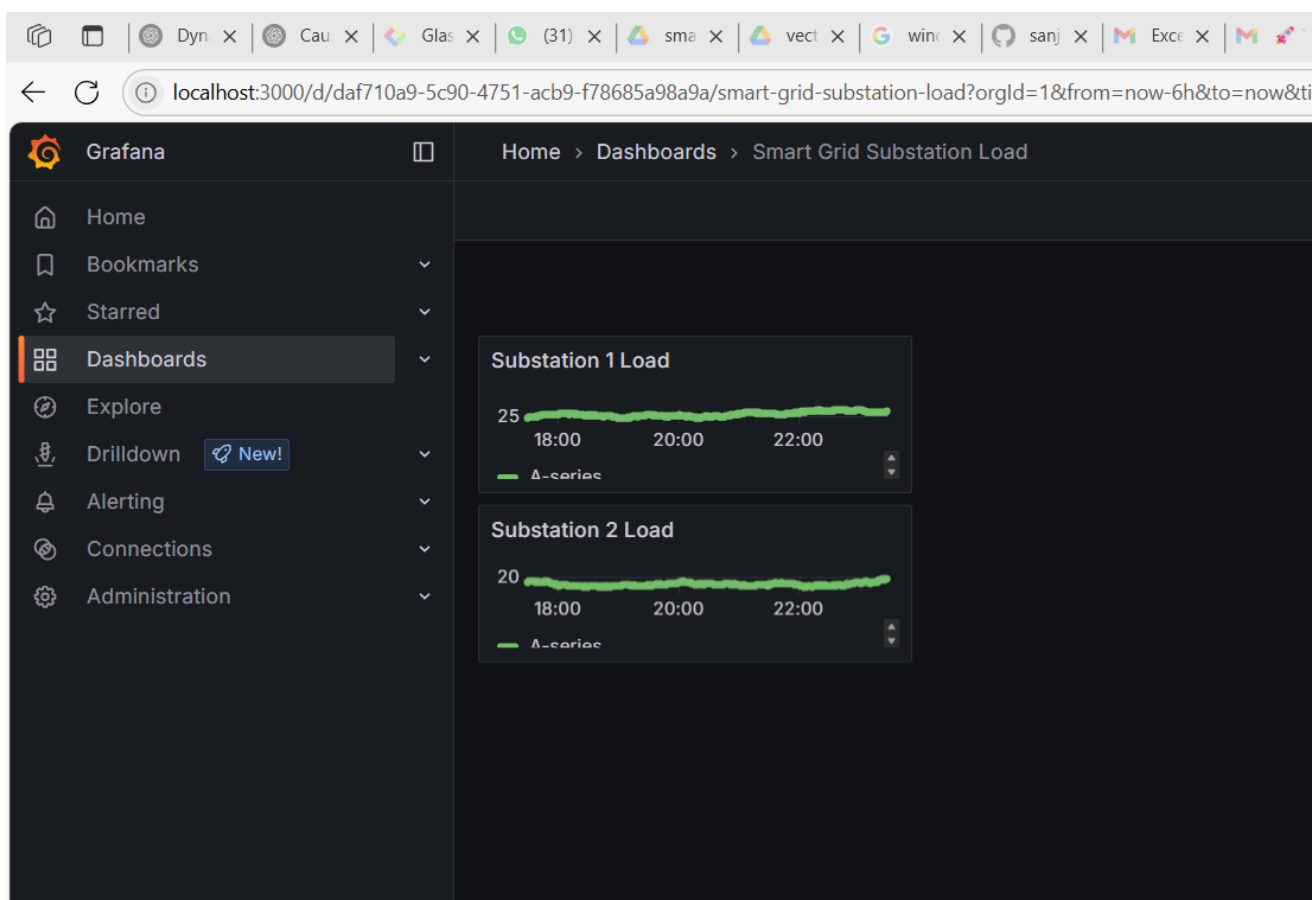

```

grafana-1 | logger=plugin.backgroundinstaller t=2025-06-24T18:42:01.490770104Z level=info msg=
tion=5.503237943s
grafana-1 | logger=infra.usagestats t=2025-06-24T18:43:01.702952834Z level=info msg="Usage sta

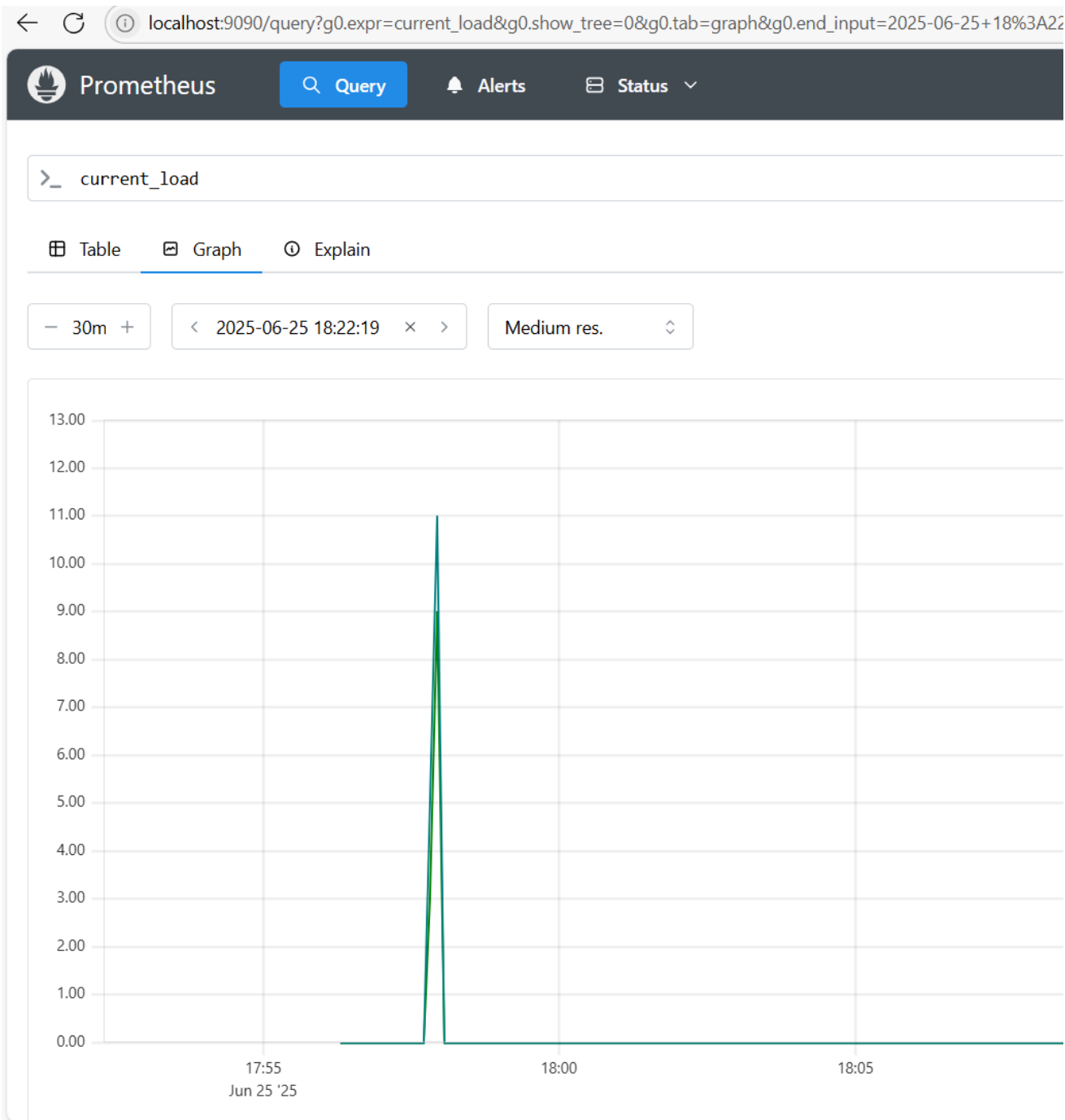
substation1-1 | 172.20.0.6 - - [24/Jun/2025 18:44:54] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.7 - - [24/Jun/2025 18:44:54] "POST /route HTTP/1.1" 200 -
charge_request-1 | 172.20.0.1 - - [24/Jun/2025 18:44:54] "POST /request_charge HTTP/1.1" 200 -
substation2-1 | 172.20.0.6 - - [24/Jun/2025 18:44:55] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.7 - - [24/Jun/2025 18:44:55] "POST /route HTTP/1.1" 200 -
charge_request-1 | 172.20.0.1 - - [24/Jun/2025 18:44:55] "POST /request_charge HTTP/1.1" 200 -
substation1-1 | 172.20.0.6 - - [24/Jun/2025 18:44:55] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.7 - - [24/Jun/2025 18:44:55] "POST /route HTTP/1.1" 200 -
charge_request-1 | 172.20.0.1 - - [24/Jun/2025 18:44:55] "POST /request_charge HTTP/1.1" 200 -
substation2-1 | 172.20.0.6 - - [24/Jun/2025 18:44:55] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.7 - - [24/Jun/2025 18:44:55] "POST /route HTTP/1.1" 200 -
charge_request-1 | 172.20.0.1 - - [24/Jun/2025 18:44:55] "POST /request_charge HTTP/1.1" 200 -
substation1-1 | 172.20.0.6 - - [24/Jun/2025 18:44:55] "POST /charge HTTP/1.1" 200 -
load_balancer-1 | 172.20.0.7 - - [24/Jun/2025 18:44:55] "POST /route HTTP/1.1" 200 -
charge_request-1 | 172.20.0.1 - - [24/Jun/2025 18:44:55] "POST /request_charge HTTP/1.1" 200 -
substation2-1 | 172.20.0.6 - - [24/Jun/2025 18:44:56] "POST /charge HTTP/1.1" 200 -

```

Prometheus output



Grafana output



Our load balancer polls metrics exposed by each substation on /metrics and dynamically routes new requests to the one with the lowest load.”

All microservices are written in Python and containerized using Docker.”

We’ve demonstrated a working, observable, and scalable microservice-based smart grid simulation.